Giugno 1997

F. Spagna

INSTALLAZIONE DI UN DATABASE RELAZIONALE SQL SU UN SERVER WEB UNIX DI ERG-FISS

F. Spagna

INSTALLAZIONE DI UN DATABASE RELAZIONALE SQL SU UN SERVER WEB UNIX DI ERG-FISS

1. INTRODUZIONE

1.1 Premessa

Presso ERG-FISS si e' presentata la necessita' di installare un database relazionale in un server Web per permettere ai client l'accesso ai dati resi ad essi disponibili.

La scelta del database e' caduta su **Mini SQL**, piu' semplicemente chiamato **mSQL**, che e' un motore di database leggero, con accesso veloce ai dati e non eccessive richieste di memoria, il cui autore e' David J.Hughes della Bond University in Australia.

1.2 Caratteristiche del database adoperato

L'interfaccia di interrogazione mediante query dell'mSQL e' un sottoinsieme dell'SQL, conforme alle specifiche ANSI SQL.

mSQL e' un database relazionale, che permette quindi relazioni tra tabelle multiple, anche se non supporta tutte le possibilita' relazionali (*joins*, *views*, *nested queries*).

Il motore database e' stato testato su piattaforme Sun (SunOS e Solaris), Ultrix, Linux e OSF/1. Pero' puo' andare anche sulla maggior parte di sistemi derivati dal BSD e dal SVR4, o sistemi POSIX. Siccome mancava ogni informazione sull'utilizzazione su IBM-RISC6000 con sistema operativo AIX si e' cercato sul Web (mailing list, vedi paragrafo 8.3) di sapere se altri lo adoperano su questa piattaforma ed hanno risposto Jens-Peter Haack del NetCS Informationstechnik GmbH di Aschaffenburg in Germania (*Peter@NetCS.com*), che ci ha comunicato che loro stanno usando mSQL su Power PC con i sistemi operativi AIX 3.2.5 e AIX 4.1.4, e *trevor@cyberspc.mb.ca*, che ci informa che il databse va molto bene su AIX versione 4. Lo stesso autore, Hughes (*bambi@Bond.edu.au*), da noi consultato, ha risposto personalmente confermando che il software puo' girare su AIX, anche se per questo sistema operativo loro non l'hanno testato.

Il pacchetto mSQL comprende:

- il motore di database, per avviare o spegnere il database server
- un programma di amministrazione di database, per gestire il database
- un programma "monitor" di terminale, per inviare le query al motore di database
- un visore di schemi, per vedere la struttura del database
- un "dumper", per creare da un database uno script adatto alla riproduzione dello stesso database
- un'interfaccia di programmazione API per l'accesso al database da programmi in linguaggio C

Il motore di database e l'API per C sono progettati per un funzionamento in ambiente client/server in una rete TCP/IP.

Il motore del database comprende un "demone", chiamato **msqld**, che e' in ascolto continuo di eventuali connessioni su un socket TCP. Esso accetta connessioni multiple e serializza le query ricevute.

1.3 Controllo di accesso

Il controllo di accesso viene gestito dal file **msql.acl** presente nella directory di installazione del database, che normalmente e' la /usr/local/Minerva. Su tale file vengono scritti i riferimenti per ogni database da controllare.

Esempio di file acl:

`database=test`	(nome del database cui ci si riferisce)
`read=spagna, meneghini, -*`	(accesso in lettura negato a tutti, salvo quelli specificati)
`write=root`	(accesso in scrittura permesso solo alla root)
`host=*.arcoveggio.enea.it`	(qualunque host nel dominio arcoveggio.enea.it)
`access=local,remote`	(accesso da connessioni locali o remote)

Se un tale file non esiste o se mancano i riferimenti ad un certo database l'accesso alla lettura ed alla scrittura e' globale.

2. COMANDI SQL DISPONIBILI

Si riportano qui i comandi (clause) SQL disponibili per l'accesso ai dati del database. I comandi sono stati evidenziati in caratteri maiuscoli, ma cio' non e' obbligatorio nella scrittura delle query, che non sono *case sensitive*.

2.1 Clause CREATE

Tale comando crea una tabella.

Esempio:

```
CREATE TABLE impiegati (cognome     char(16),
                        nome        char(16) not null,
                        numero      int primary key,
                        matricola   int )
```

dove i tipi possibili sono: **`char(lungh)`** (stringa di caratteri di una determinata lunghezza), **`int`** (interi con segno di 4 byte) e **`real`** (valori reali in notazione decimale o scientifica). Un campo puo' essere definito o no come chiave primaria (`primary key`) o `not null`.

2.2 Clause DROP

Tale comando cancella una tabella da un database.

Esempio:

```
DROP TABLE impiegati
```

2.3 Clause INSERT

Questo comando inserisce dei dati.

Esempio:

```
INSERT INTO impiegati (cognome, nome, matricola)
VALUES ('Spagna', 'Ferruccio', 79960)
```

2.4 Clause DELETE

Tale comando cancella una riga.

Esempio:

```
DELETE FROM impiegati  WHERE numero = 123
```

Per la condizione `WHERE` vedi paragrafo seguente.

2.5 Clause SELECT

Questo e' il comando SQL fondamentale per fare delle ricerche mediante una selezione dei dati che soddisfano determinate condizioni.

Esempio:

```
SELECT [DISTINCT] cognome, nome FROM impiegati
WHERE matricola = 79960 AND cognome LIKE '_pagna'
ORDER BY Cognome, nome DESC
```

ove:

SELECT	seleziona i campi
FROM	indica da quale tabella vengono selezionati i dati
WHERE	esprime una condizione (=, <, >, <=, >=, <>, LIKE)
AND	somma due condizioni
LIKE	e' una condizione con certi caratteri indeterminati (vedi sotto per _ e %).
ORDER BY	dice se i dati ordinati per ordine decrescente (se DESC, altrimenti crescente)
DISTINCT	se si vogliono eliminare le righe duplicate

I caratteri speciali sono:

'_' indica un carattere qualunque (in una condizione LIKE)

'%' indica un numero qualsiasi di carattere di qualunque valore (in una condizione LIKE)

'\' viene fatto precedere ai caratteri speciali: per es. \% equivale a % e \\ equivale a \

Il comando:

```
SELECT *  etc.
```

si riferisce a tutti i campi delle righe.

2.6 Clause UPDATE

Tale comando aggiorna un campo.

Esempio:

```
UPDATE impiegati SET matricola = 79960 WHERE numero = 123
```

2.7 SCRIPT DI QUERY SQL

I comandi per le query SQL possono essere scritti su un file di testo (script) che potra' essere dato come input al motore database (vedi paragrafo 6.2). Un esempio di script e' riportato qui:

```
# ------------------------------
# Esempio di file script miniSQL
# ------------------------------

drop table tabella1                    \p\g
drop table tabella2                    \p\g

create table tabella1 (
   nome   char(10) primary key,
   matr   int,
   tel    char(20),
```

```
   indir  char(20) )                    \p\g

create table tabella2 (
   user   char(10) primary key,
   nome   char(30) )                    \p\g

insert into tabella1(nome, matr, tel) values ('Spagna', 36, '3488')  \p\g
insert into tabella1(nome, matr, tel) values ('Meneghini', 35, '3411')
\p\g
insert into tabella2(user, name) values('Spagna', 'F. Spagna')       \p\g
insert into tabella2(user, name) values('Meneghini', 'L. Meneghini')
\p\g


# ------------
# Keyed lookup
# ------------

select tel from tabella1 where name = 'Spagna'  \p\g
select * from tabella1                          \p\g


# ------------------
# Paul's birthday :-)
# ------------------

update tabella1 set matr=46 where name='Spagna' \p\g
select * from tabella1                          \p\g


# ---------------------
# Non-key passed lookup
# ---------------------

select * from tabella1 where tel = '3488'       \p\g
select * from tabella1 where name like '_pag%'  \p\g


# ---------
# Do a join
# ---------

select tabella2.nome, tabella1.tel
      from tabella,tabella1
      where tabella2.user = tabella1.nome       \p\g

# ----------------
# Try a sorted one
# ----------------

select tabella2.name, tabella1.tel
      from tabella2,tabella1
      where tabella2.user = tabella1.nome
      order by tabella2.nome                    \p\g
```

2.8 Presentazione dei risultati di una query

I risultati di una query, ad esempio:

```
$ msql elenco
msql > select cognome, tel from telefoni where cognome like 'SP%' \g
```

sono presentati sullo schermo in una forma del tipo:

```
Query OK.
3 rows matched.
+----------------------+----------+
| cognome              | tel      |
+----------------------+----------+
| Speranza             | 3568     |
| Spagna               | 3488     |
| Spadoni              | 3514     |
+----------------------+----------+
```

3. IL MOTORE DI DATABASE

Il "demone" mSQL e' un'applicazione che, una volta lanciata con il comando:

```
$ msqld
```

resta in attesa di connessioni su un determinato socket TCP (accesso remoto in rete) o un socket di dominio UNIX. Esso supporta connessioni multiple e serializza le query ricevute.

4. AMMINISTRAZIONE DEL DATABASE

I database mSQL sono amministrati mediante un comando della forma:

```
$ msqladmin operazione
```

ad esempio:

```
$ msqladmin create mioDB
```

mediante il quale si possono fare (soltanto il *root* ne e' autorizzato) alcune operazioni come creare nuovi database o chiudere il server. Il comando richiede necessariamente come argomento uno dei comandi disponibili, che sono:

`create nomeDataBase`	(crea un nuovo database)
`drop nomeDataBase`	(cancella un intero database esistente)
`shutdown`	(spegne il server)
`reload`	(dice al server di ricaricare l'informazione di controllo d'accesso)
`version`	(fornisce informazioni sulla versione)

Il comando accetta il flag di linea di comando `-hHost` per specificare la macchina.

5. MONITOR DI TERMINALE mSQL

5.1 Query da linea di comando

Il programma monitor che permette di sottoporre interattivamente da linea di comando delle query ad un motore database mSQL si chiama con il comando:

```
$ msql nomeDatabase
```

che richiede necessariamente come argomento il nome del database. Il programma e' stato fatto sul modello di quello del monitor del database Ingres. Il lancio del comando introduce nell'ambiente di query caratterizzato dal prompt:

```
msql >
```

I comandi al monitor sono costituiti da certi caratteri preceduti da un \ (backslash), ad esempio **\h** (h sta per help) da' l'help e **\q** (q sta per quit) fa uscire dal programma (l'uscita si puo' ottenere anche con Ctrl-D).

Le query da inviare al motore mSQL possono essere sottoposte in vario modo: in primo luogo da linea di comando direttamente, come ad esempio:

```
msql > update miaTabella set telefono = 3488 where nome = 'Spagna'  \g
```

dove il comando **\g** (g sta per Go), scritto alla fine, produce l'effetto di inviare la query che lo precede. Cio' che viene scritto in linea di comando viene posto, dopo il tasto invio, in un buffer di query. Se la query ha un carattere `\g` alla fine, essa viene inviata subito al monitor con il tasto invio, altrimenti resta sul buffer e puo' essere inviata successivamente con un comando `\g` separato in linea di comando:

```
msql >  \g
```

Una o piu' query possono essere inviate al monitor dopo averle redatte con un editore di testo: il comando:

```
msql >  \e
```

ove il comando **\e** (e sta per edit) permette di editare la query precedente o una nuova se non ce n'e' alcuna nel buffer, aprendo automaticamente l'ambiente **vi**, o altro editore se se ne e' scelto uno diverso, uscendo dal quale la query e' inviata al buffer e subito eseguita se essa termina con il comando `\g`.

La memorizzazione su buffer dell'ultima query permette il rinvio della stessa query successive volte con diversi comandi `\g` oppure l'eventuale riedizione di essa con `\e`.

5.2 Query da file script

Un altro modo di sottoporre delle query e' quello di scrivere le query su un file script, file portante l'estensione .msql, e poi sottoporre il file al programma con il comando:

```
$ msql nomeDB<fileScript.msql
```

Il contenuto del buffer di query puo' essere visualizzato con il comandi **\p** (p sta per print).

Il comando `msql` accetta due tipi di flag sulla linea di comando:

`-hHost`   per una connessione al server mSQL su host

-q per uscire (quit) dopo l'esecuzione della query.

6. VISORE DI SCHEMA

Il comando **relshow** permette di visualizzare la struttura di un database. Se il comando e' usato da solo, senza argomenti:

```
$  relshow
```

esso mostrera' il nome di tutti i database disponibili cosi':

```
+-----------------+
|    Databases    |
+-----------------+
|    Database1    |
+-----------------+
```

Se invece `relshow` e' usato con argomento uguale al nome di un database esistente:

```
$  relshow nomeDB
```

sono visualizzate le tabelle di quel database nella forma:

```
+-----------------+
|      Table      |
+-----------------+
|    Tabella1     |
+-----------------+
```

Se infine `relshow` viene usato con il nome di un database e quello di una tabella:

```
$  relshow nomeDB nomeTabella
```

sara' visualizzata la struttura della tabella con i nomi dei campi, il loro tipo e le loro dimensioni, con uno schema della forma seguente:

```
+----------------+---------+---------+----------+---------+
|     Field      |  Type   | Length  | Not Null |   Key   |
+----------------+---------+---------+----------+---------+
|     nome       |  char   |   16    |    N     |    N    |
+----------------+---------+---------+----------+---------+
|   matricola    |  int    |    4    |    N     |    N    |
+----------------+---------+---------+----------+---------+
```

Il comando `relshow` accetta il flag di linea di comando `-h Host` con cui si specifica una macchina remota come server del database.

7. DATABASE DUMPER

Il programma chiamato con:

```
$  msqldump nomeDB [nomeTabella]
```

scarica il contenuto e la struttura di una tabella o di un intero database in un file in formato testo con i comandi opportuni adatto per essere letto dal monitor come file script per riprodurre il database o la tabella, secondo quanto spiegato al paragrafo 6.2.

8. LIBRERIA (API) DI PROGRAMMAZIONE IN C

8.1 Uso dell'API in C

La libreria di API per il C permette ad un programma scritto in linguaggio C di comunicare con il motore di database. Un programma di questo tipo deve includere lo header **msql.h** ed essere linkato con la libreria **libmsql.a** (argomento `-lmsql` alla compilazione su riga di comando). La libreria rende disponibili le funzioni presentate nei paragrafi seguenti. Per default msql.h e' installato nella directory /usr/local/Minerva/include e msql.a in /usr/local/Minerva/lib.

8.2 Funzione mqslConnect()

```
int msqlConnect(char *host)
```

Questa funzione stabilisce una connessione con un motore mSql. Richiede come argomento l'indirizzo IP dell'host che gestisce il server mSQL. Se tale argomento e' posto uguale a `NULL` la connessione e' fatta ad un server che funziona su host locale usando il socket di dominio UNIX (/dev/msqld ?).

Tale funzione restituisce un intero che rappresenta il ***socket descriptor*** della connessione, che puo' essere usato come *handle* per le successive chiamate di funzioni dell'API che richiedono come argomento il socket. Se c'e' un errore oppure se non si e' potuta stabilire la connessione il valore restituito e' -1.

Sono possibili diverse connessioni contemporanee a diversi server di database con diverse chiamate di questa funzione.

8.3 Funzione msqlSelectDB()

```
int msqlSelectDB(int socket, char *nomeDB)
```

Questa funzione seleziona il database a cui si vuole accedere per effettuarvi delle query, e in essa il primo argomento e' il *socket descriptor* restituito dalla funzione mqslConnect() ed il secondo e' il nome del database. E' possibile chiamare diverse volte questa funzione anche con parametri diversi: ciascuna volta il server, passando da un database all'altro, usera' per gli accessi successivi il database specificato.

8.4 Funzione mqslQuery()

```
int msqlQuery(int socket, char *query)
```

Questa funzione, che ha come primo argomento il *socket descriptor* restituito dalla funzione `mqslConnect()`, invia una query come stringa di testo al motore del database sulla connessione associata con il socket specificato.

Se la query ottiene dei risultati dal motore di database (come in seguito ad una query di tipo `SELECT`), i dati sono posti in un buffer e possono quindi essere salvati con la funzione `msqlStoreResult()`. Se non vengono salvati, i dati saranno sovrascritti dalle query seguenti.

8.5 *Funzione msqlStoreResult()*

```
m_result *msqlStoreResult()
```

Questa funzione, che restituisce un puntatore ad una struttura di tipo `m_result`, salva i dati generati da una query di tipo `SELECT`. E' un'operazione da fare prima di chiamare una nuova query per non perdere i dati residenti sul buffer, che altrimenti verrebbero soprascritti. La funzione restituisce un puntatore ad una struttura di tipo `m_result` e sara' l'oggetto cosi' creato ad essere utilizzato come parametro nelle funzioni msql che seguono in questa lista, che devono quindi essere sempre precedute dal salvataggio dei dati con la funzione `mqslStoreResult()`. Di questi *result handle* ce ne possono essere diversi contemporaneamente in un programma.

8.6 *Funzione msqlFreeResult()*

```
void msqlFreeResult(m_result *result)
```

Questa funzione puo' essere chiamata quando i dati generati da una query e salvati in un *result handle* non sono piu' necessari, per liberarli.

8.7 *Funzione msqlFetchRow()*

```
m_row msqlFetchRow(m_result *result)
```

La funzione, che permette l'accesso alle righe del database risultanti da un `SELECT`, restituisce una struttura di tipo `m_row` che contiene un puntatore a carattere per ognuno dei campi selezionati della riga. Un valore `NULL` e' restituito quando e' stata raggiunta la fine dei dati.

8.8 *Funzione msqlDataSeek()*

```
void msqlDataSeek(m_result *result, int posizione)
```

La struttura `m_result` possiede una variabile "cursore" che da' il numero della prossima riga. La funzione `msqlDataSeek()` permette di cambiare la posizione del cursore, dando il valore come parametro (posizione), essendo 0 corrispondente alla prima riga. Se il parametro posizione oltrepassa la dimensione della tabella la funzione restituisce `NULL`.

8.9 *Funzione msqlNumRows()*

```
int msqlNumRows(m_result *result)
```

Questa funzione restituisce il numero di righe risultanti da una query. L'argomento da passare e' l'*handle* di tipo `m_result` restituito da `msqlStoreResult()`. La funzione restituisce il numero di righe risultante dalla query (che puo' eventualmnte essere anche 0).

8.10 *Funzione msqlFetchField()*

```
m_field msqlFetchField(m_result *result)
```

Questa funzione fornisce le caratteristiche di ciascuno dei campi (uno alla volta, NULL quando i campi sono terminati) della tabella mediante un puntatore ad una struttura di tipo `m_field` cosi' definita:

```
typedef struct {
   char *nomeCampo,
        *nomeTabella;
   int  tipoCampo,
        lunghCampo,
        flagsAttributo;
} m_field;
```

8.11 Funzione msqlFieldSeek()

```
void msqlFieldSeek(m_result *result, int pos)
```

Una struttura di tipo `m_result` contiene come membro la posizione del cursore per i campi. La funzione `msqlFieldSeek()` puo' fissare la posizione di tale cursore.

8.12 Funzione msqlNumFields()

```
int *msqlNumFields(m_result *result)
```

Questa funzione fornisce il numero di campi restituiti da una query. Questo valore rappresenta il numero di elementi del vettore di dati ritornato dalla funzione `msqlFetchRow()`. E' buona norma chiamare tale funzione per poi evitare di oltrepassare il limite del vettore di dati.

8.13 Funzione msqlListDBs()

```
m_result *msqlListDBs(int socket)
```

Tale funzione da' una lista dei database conosciuti dal motore mSQL restituendo un puntatore a strutture di tipo `m_result`.

8.14 Funzione msqlInitDB()

```
msqlInitDB()
```

Seleziona un database.

8.15 Funzione msqlListTables()

```
m_result *msqlListTables(int socket)
```

Questa funzione fornisce una lista delle tabelle definite in un database selezionato con una funzione `msqlInitDB()`, restituendo un *result handle*.

8.16 Funzione msqlListFields()

```
m_result *msqlListFields(int socket, char *nomeTabella)
```

Tale funzione fornisce i campi in una particolare tabella.

8.17 Funzione msqlClose()

```
int msqlClose(int socket)
```

Funzione che chiude la connessione con il motore mSQL. L'argomento della funzione e' il *socket descriptor* restituito dalla `msqlConnect()` al momento dell'apertura della connessione.

. LIBRERIA (API) DI PROGRAMMAZIONE IN JAVA

!.1 *Uso dell'API in Java*

MsqlJava e' una libreria di classi Java che permette ad applicazioni Java o ad applet di accedere manipolare database mSQL. Si trova in:

http://mama.minmet.uq.oz.au/msqljava

).2 *Classe Msql*

La classe **Msql** possiede i metodi seguenti.

).2.1 Metodo Msql()

```
Msql()
```

Metodo constructor per l'istanziazione di un oggetto Msql.

9.2.2 Metodo Connect()

```
Connect()
```

Questo metodo puo' avere uno, due o tre argomenti (*overloaded*).L'argomento del metodo e' la stringa rappresentante il nome dell'host che gestisce il server mSQL (si assume che la porta di destinazione sia la radice, cioe' la 1112). Il metodo apre un socket sulla destinazione data. Il server invia al client come stringa la versione del server ed il client invia quindi al server il nome dell'utilizzatore, perche' il server ne possa fare i controlli per eventuali restrizioni mediante l'ACL (Access Control List, vedi paragrafo 1.3). Il server invia infine al client una stringa di stato indicante se l'utilizzatore ha avuto accesso. Se non sono sorte eccezioni a questo punto viene stabilita la connessione.

9.2.3 Metodo SelectDB()

```
SelectDB()
```

Vedi funzione analoga nell'API in C.

9.2.4 Metodo ListFields()

```
MsqlFieldDesc ListFields()
```

Vedi funzione analoga nell'API in C.

).2.5 Metodo Query()

```
MsqlResult Query()
```

Tale metodo viene usato per inviare le query al server e restituisce un oggetto di classe `IsqlResult` (vedi paragrafo 10.3), che rappresenta il risultato della query (solo quelle di tipo `;ELECT` producono dei dati).

).3 *Classe MsqlResult*

).3.1 Membri della classe

La classe **`MsqlResult`** serve a contenere e trattare i dati risultanti da query di tipo `SELECT`. Tali query producono una nuova tabella che e' contenuta nell'oggetto restituito dal metodo, ma ust'oggetto comprende anche una seconda tabella contenente informazioni sulle colonne presenti iella prima tabella. La classe `MsqlResult` possiede i metodi seguenti.

).3.2 Metodo FetchRow()

```
String[] FetchRow()
```

Vedi funzione analoga nell'API in C.

9.3.3 Metodo Close()

```
Close()
```

Metodo che chiude la connessione al server mSQL.

9.3.4 Istruzioni fondamentali

Ogni programma Java per mSQL possiede in generale le seguenti istruzioni fondamentali:

```
Msql M = new Msql();                                    // istanzia l'oggetto
M.Connect("mioSito");                               // stabilisce la connessione
M.SelectDB("mioDB");                                   // seleziona il database
MsqlResult R = M.Query("select... from miaTab..."); // esegue la query
M.Close();                                             // chiude la connessione
```

9.3.5 Package msql

Il **package msql** e' disponibile presso:

http://mama.minmet.uq.oz.au/msqljava/packages.html

Esempio di programma in Java

1 Esempi dell'autore di MsqlJava

'ella pagina Web *htt://mama.minmet.uq.oz.au/msqljava/msqlexamples.html* sono riportati i tre ıpi seguenti di Darryl Collins (*darryl@minmet.uq.oz.au*), con la messa a disposizione del :e sorgente:

1sqlDemo.java

1sqlJavaStats.java

1sqlViewer.java

.2 Esempio nostro

;i riporta qui un esempio di un semplice programma scritto da noi in Java per l'accesso ad un ıbase mSQL.

ACCESSO DA LINGUAGGIO DI SCRIPT PERL

Per l'accesso all'mSQL mediante Perl e' disponibile la Perl 5 Interface in:

ftp://Bond.edu.au/pub/Minerva/msql/Contrib/DBPerl/

nei file:

DBD-msql-0.59.tar.Z e

DBI-0.58.tar.Z

. PROGRAMMA CON INTERFACCIA HTML A mSQL

Un programma di Sol Katz (*skatz@blm.gov*), scritto per creare un'interfaccia html a mSQL e' ;ponibile sul Web sotto forma del file:

ft://ftp.blm.gov/pub/gis/msqlc2.zip

mprendente i file cgi in C. Tutte le informazioni relative al database ed alle tabelle da usare sono ·ste nell'html, e pertanto il programma non deve essere modificato quando si usano altri database .abelle.

2. IMPORTAZIONE DI DATI DA UN FILE DI TESTO

Il programma **msql-import** (autore Pascal Forget, *pascal@wsc.com*), scritto in linguaggio C, :rmette di importare in una tabella mSQL dei dati scritti su un file di testo e separati da un :limitatore qualsiasi in una tabella di un database. Il programma non crea la tabella, che deve ;sere stata creata prima, e agisce inviando delle query di tipo INSERT al server mSQL. 'isponibile come:

ftp://Bond.edu.au/pub/Minerva/msql/Contrib/msql-import-0.0.9.tar.gz

3. ESEMPIO DI IMPORTAZIONE DI UN DATABASE

L'esempio che si riporta in questo paragrafo consiste nell'importazione in un database mSQL di ati da un database esistente in formato dBase (.dbf) attualmente usato dall'amministrazione del entro per i numeri telefonici dei dipendenti.

La prima operazione e' quella di **creazione del database**, che chiamiamo "elenco", ed e' ffettuata dal *root* sulla macchina Risc365, su cui e' installato l'mSQL, mediante il comando:

```
$ msqladmin create elenco
```

Viene quindi editato (da parte del *root*) il file msql.acl (vedi paragrafo 1.3) situato in ısr/local/Minerva per aggiungervi le righe relative al **controllo di accesso** al nuovo database:

```
database = elenco
read = *
write = spagna, root, -*
host = *.arco.e.it
access = local,remote
```

Siccome poi si vuole importare la tabella del database originario che ha la struttura seguente:

cognome	20	T
nome	20	T
matr	5	T
tel		N
segr		N
fax		N
sede	8	T
sottosede	2	T
unita	20	T
pal	2	T
stanza	4	T
titolo	5	T

sottopone la query seguente per **creare la struttura** della nuova tabella mSQL:

```
Strutture della tabella 'telefoni'

EATE TABLE telefoni (
 cognome CHAR(20),
 nome CHAR(20),
 matr CHAR(5),
 tel INT,
 segr INT,
 fax INT,
 sede CHAR(8),
 sottosede CHAR(2),
 unita CHAR(20),
 pal CHAR(2),
 stanza CHAR(4),
 titolo CHAR(5)
 \g
```

A questo punto, essendo cosi' creati il database e la tabella con la sua struttura, si puo' lavorare l'importazione, per la quale ci si serve del programma DBMS Microsoft Access (ma si sarebbe uto utilizzare qualunque altro sistema analogo, per esempio l'importazione con Microsoft Excel mette un trattamento di testo piu' agevole). Il file dBase, chiamato tel4.dbf, viene importato in

cess come database. Siccome si vuole quindi preparare uno **script di query** per introdurre i :ord di tab4.dbf con una serie di righe del tipo:

```
SERT INTO telefoni VALUES ('ABBATTISTA', 'MARIATERESA', '1048', 3403,
  0,  0,  'BOLOGNA',  'A',  'ERG-BOL-SEPER',  'F',  '109',  'Sig.a')

ISERT INTO telefoni VALUES ('ADORNI',     'GIOVANNA',    '1308', 3758,
  0,  0,  'BOLOGNA',  'A',  'HPCN',                  'C',  '301',  'D.ssa')
r
```

aggiungono in ambiente Access (nel quale tali operazioni sono agevoli) alla tabella altri campi e rappresentano l'inizio del rigo (un primo campo e' costituito dalla stringa da "*INSERT...* fino *..VALUES (*' "), i separatori tra campo e campo (tipo *[','] [',]* o *[,']*) e la fine del rigo (ultimo mpo costituito da *[') \g]*). Quest'operazione va fatta precedere da una sostituzione di tutti gli ıostrofi, gia' presenti nella tabella originaria come accenti, con caratteri \ '.

Si puo' allora esportare il database come testo su un file ASCII cui si da' il nome tel4.msql, che gia' adatto per essere sottoposto come **script** al motore msql mediante il comando:

```
$ msql elenco<tel4.msql
```

La tabella e' cosi' riempita con i dati voluti e l'operazione e' conclusa.

4. DISPONIBILITA' DEL SOFTWARE

4.1 *Disponibilita' dell'mSQL*

Una **Mini SQL Home Page** e' presentata sul sito Web Hughes Technologies, all'URL:

http://Hughes.com.au/product/msql/

ı cui viene fatta una prima presentazione di mSQL e da cui si possono richiamare i seguenti ocumenti:

- Frequently Asked Questions (FAQ) (faq.htm) (gestito da *Peter.Samuel@uniq.com.au*)
- Conference Paper (paper.htm)
- On-Line Manual (manual.htm)
- Release History (history.htm)

collegarsi al sito ftp da cui fare il **downloading** del software e cioe':

ftp://Bond.edu.au/pub/Minerva/msql

lella Bond University, Gold Coast, Queensland, Australia, sul quale e' disponibile la versione di nSQL costituita dal file compresso **msql-1_0_15_tar.gz** (191 KB), che e' la versione del 3 luglio)6, da noi utilizzata per le prove, e i documenti **faq.html** e **faq.txt**.

14.2 *Disponibilita' dell'MsqlJava*

Nella directory `Contrib/` del sito ftp suddetto sono disponibili per il downloading diverse interfacce all'mSQL da Java, Perl, Tcl, REXX e Python. Noi abbiamo scaricato i file relativi all'interfaccia Java (autore Darryl Collins):

- MsqlJava-1_1_1.zip (22 KB)

- dbview.c (16KB)
- dbview.c.lsm (563 B)
- dbview.README (1 KB)
- msql-JDBC_0_9_3_tar.gz (32 KB) driver JDBC per mSQL di Darryl Collins.

Presso il sito:

http://mama.minmet.uq.oz.au/msqljava/

(riportato anche da *http://www.gamelan.com*) si possono trovare i documenti:

- Tutorial
- FAQ
- Inside MsqlJava
- Documentation
- Download MsqlJava (stesso sito gia' riportato sopra)
- Examples: tre esempi di programmi Java per accesso ad alcuni database presenti sul sito.

Degli esempi di uso di Java per l'mSQL sono riportati all'URL:

http://adams.patriot.net/~anil/java/javaSQL/

4.3 Installazione dell'mSQL sull'IBM-RISC6000

Il file msql-1_0_15_tar.gz, esploso con il comando:

```
$ gunzip msql-1_0_15_tar.gz
```

:nera un file msql-1_0_15_tar che viene trattato con:

```
$ tar -tvf msql-1_0_15_tar
```

si vengono cosi' a creare in totale 2480 KB:

files: `Makefile`
`README` (con le istruzioni per l'installazione)

directory: `bin`
`doc` (con vari documenti di testo)
`include` (con diversi file #include)
`lib` (con diversi file di libreria)
`scripts` (con diversi file script, tra cui run_daemon per lanciare il server)
`src` (con diversi file sorgente: .C e .H in sei sottodirectory
e vari script di casi di prova)

Per fare il build dell'mSQL il comando e':

```
$ make target
```

he crea una directory target dove andranno i file object per il sistema ospitante:

TOP)/targets/AIX-4.1-rs6000

poi, da questa directory si esegue:

```
$  setup
```

uindi:

```
$  make       o make istall?
```

fine:

```
$  make all
```

La directory di default dell'installazione sara' `/usr/local/Minerva/`.

Una volta installato mSQL si puo' provare lo script di prova sample.msql.

4.4 Mailing list

Una *mailing list* disponibile per gli utenti di mSQL, cui ci siamo inscritti, e' all'indirizzo:

sql-list-request@Bunyip.com per l'iscrizione mediante "*subscribe*"

sql-list@Bunyip.com per l'invio di messaggi